

Trabalhando com Janelas Java (Swing)

Nádia Félix

nadia@inf.ufg.br

Interface Gráfica

Objetivo:

- Demostrar o funcionamento dos principais componentes do pacote javax.swing;
- Identificar os principais componentes presentes em janelas de uma aplicação em Java: botões, campos texto, barras de rolagem, lista de múltiplas escolhas, entre outros;
- Demonstrar a utilização de diversos tipos de caixa de mensagens e menus;
- Definir os principais métodos usados no controle de layout de uma aplicação por meio dos gerenciadores de layout.

Classes do Pacote javax.swing

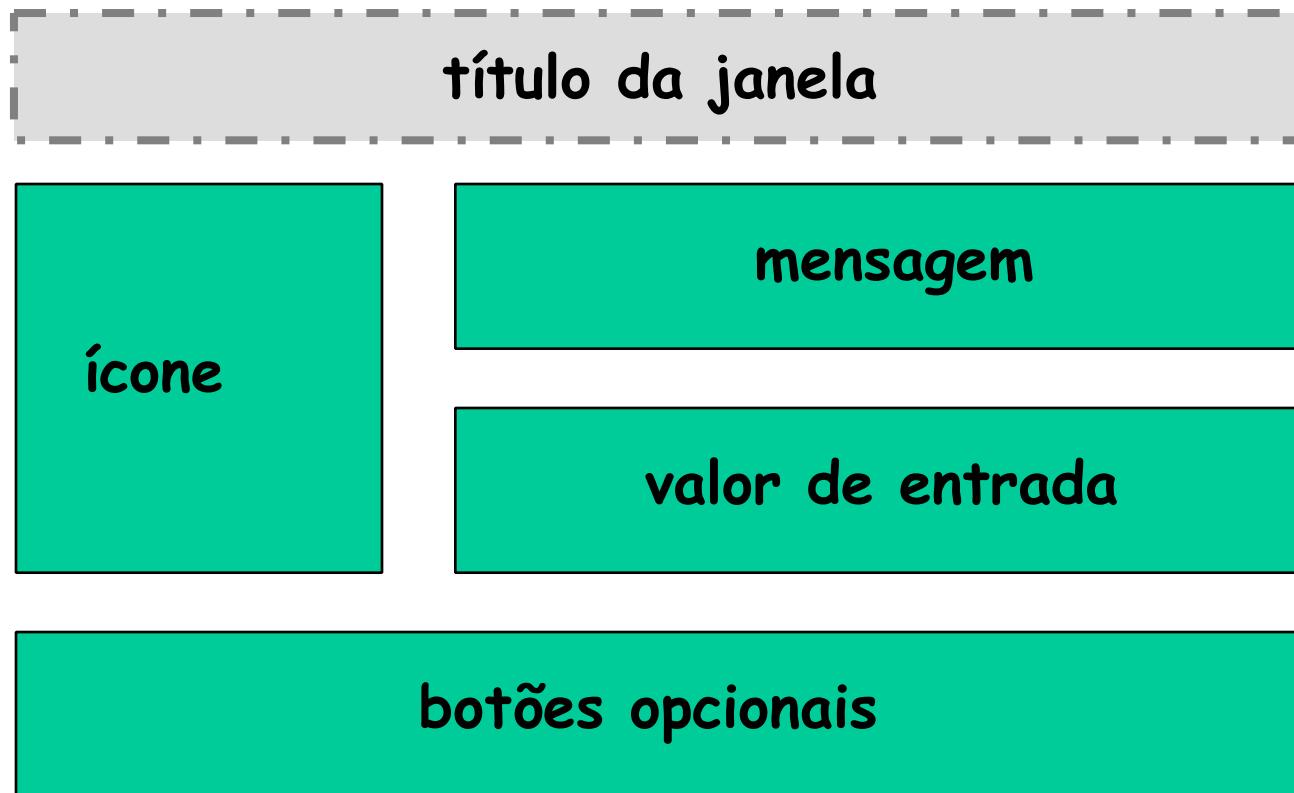
- Classes apropriadas para a criação de aplicações gráficas.
- São extensões do pacote **java.awt**, que por sua vez são extensões da *Object*.
- A utilização das classes do **javax.swing** melhorou: a aparência, o tratamento de eventos e a portabilidade.
- A diferença com relação ao nome das classes está na presença da letra J antes do início do nome. Por exemplo a classe **Button** (*java.awt*) e **JButton** (*javax.swing*).

A classe JOptionPane

- Através da classe **JOptionPane** é fácil exibir caixas de diálogo de diferentes tipos que informe algo ou recupere alguma informação do usuário.
- A classe parece ser complexa devido ao grande número de métodos, porém quase todas as chamadas são constituídas de métodos **estáticos**.
- Na sua forma mais básica, ela exibe uma linha de mensagem e um botão rotulado com “OK” e aguarda a interação do usuário.

A classe JOptionPane

- A aparência de uma das caixas de diálogo é similar à figura abaixo:

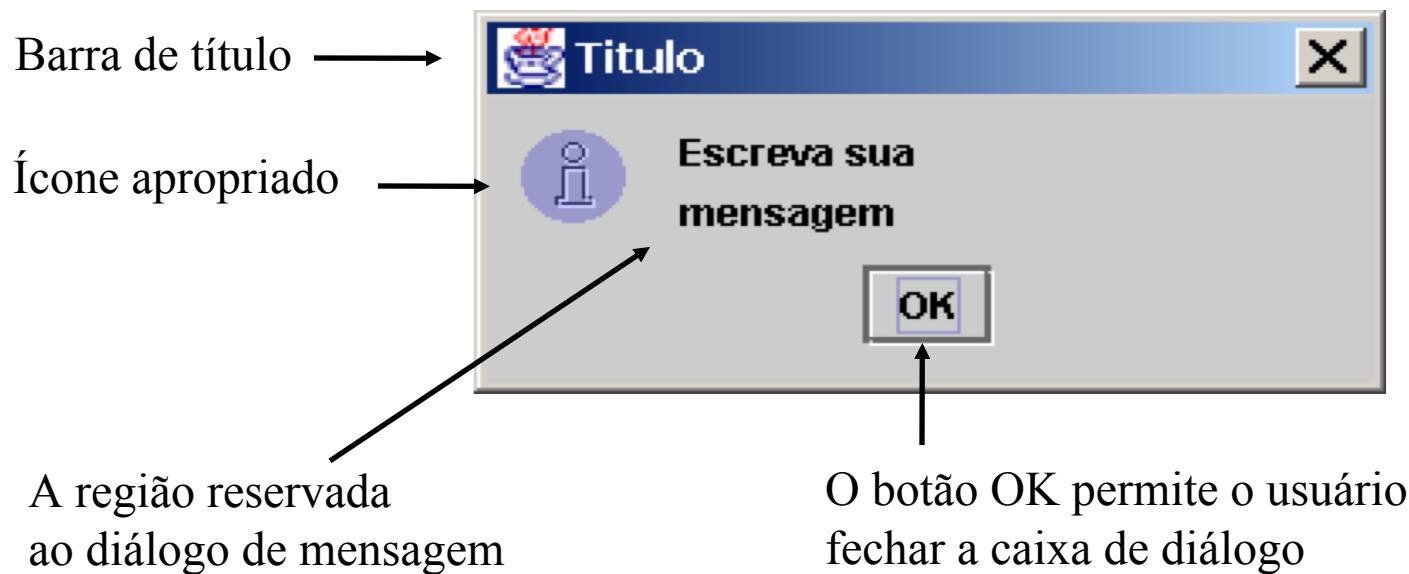


A classe JOptionPane

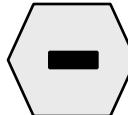
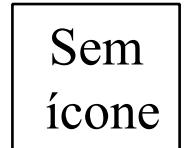
- Tipos de caixas de diálogo:
 - **MessageDialog** - mostra uma mensagem;
 - **ConfirmDialog** – mostra uma mensagem e possibilita ao usuário responder a uma pergunta;
 - **InputDialog** - mostra uma mensagem e permite a entrada de um texto;
 - **OptionDialog** - abrange os três anteriores.

Exibindo a saída em uma janela ou caixa de diálogo

- Utilizando a classe JOptionPane do pacote javax.swing.



Tipos de diálogo de mensagem

Tipo	Ícone	Descrição
JOptionPane.ERROR_MESSAGE		Exibe um diálogo que indica um erro.
JOptionPane.INFORMATION_MESSAGE		Exibe um diálogo informacional.
JOptionPane.WARNING_MESSAGE		Exibe um diálogo de advertência.
JOptionPane.QUESTION_MESSAGE		Exibe um diálogo que impõe uma pergunta ao usuário. Em botões Yes ou NO.
JOptionPane.PLAIN_MESSAGE		Exibe um diálogo somente com mensagem.

A classe JOptionPane

Constantes para tipos de mensagens:

`JOptionPane.ERROR_MESSAGE`

`JOptionPane.INFORMATION_MESSAGE`

`JOptionPane.WARNING_MESSAGE`

`JOptionPane.QUESTION_MESSAGE`

`JOptionPane.PLAIN_MESSAGE`

Constantes para opções de botões:

`JOptionPane.YES_NO_OPTION`

`JOptionPane.YES_NO_CANCEL_OPTION`

`JOptionPane.OK_CANCEL_OPTION`

Constantes inteiras de retorno:

`JOptionPane.YES_OPTION`

`JOptionPane.NO_OPTION`

`JOptionPane.CANCEL_OPTION`

`JOptionPane.OK_OPTION`

`JOptionPane.CLOSED_OPTION`

JOptionPane - Variações

Personaliza mensagem dos botões

```
Object[] options = { "OK", "CANCELA" };  
JOptionPane.showOptionDialog(null, "Continua?", "Warning",  
                           JOptionPane.YES_NO_OPTION,  
                           JOptionPane.WARNING_MESSAGE, null,  
                           options, options[0]);
```

Pré-define valores de entrada em uma comboBox

```
Object[] valoresPossiveis= { "Um", "Dois", "Três" };  
Object selectedValue = JOptionPane.showInputDialog(null,  
                                                 "Escolha um","Input", JOptionPane.INFORMATION_MESSAGE,  
                                                 •null, valoresPossiveis, valoresPossiveis[0]);
```

Exemplo MessageDialog

Sintaxe: `JOptionPane.showMessageDialog`

(Componente, mensagem, titulo, tipo da mensagem, icone)

```
import javax.swing.JOptionPane;

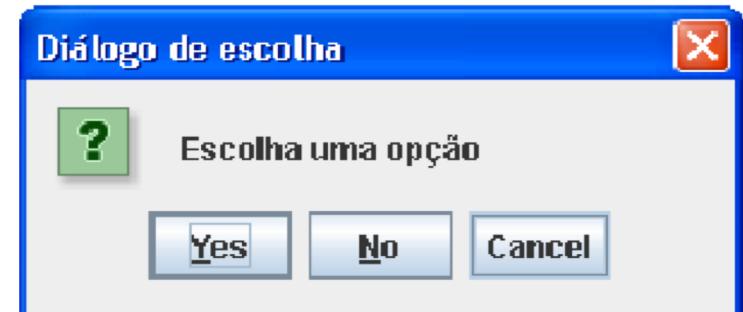
public class TesteMessageDialog {
    public static void main (String [] args)
    {
        JOptionPane.showMessageDialog(null,
            "Hello Internet gráfico!",           //Mensagem
            "Primeiro Programa Gráfico",          //Título
            JOptionPane.INFORMATION_MESSAGE);     //tipo de mensagem
        System.exit(0);
    }
}
```



Exemplo ConfirmDialog

Sintaxe: `JOptionPane.showConfirmDialog`
`(Componente, mensagem, titulo, tipo dos botões, ícone)`

```
import javax.swing.JOptionPane;  
  
public class TesteConfirmDialog {  
  
    public static void main (String [] args)  
    {  
  
        JOptionPane.showConfirmDialog(null,  
            "Escolha uma opção",  
            "Diálogo de escolha",  
            JOptionPane.YES_NO_OPTION);  
  
        System.exit(0);  
    }  
}
```



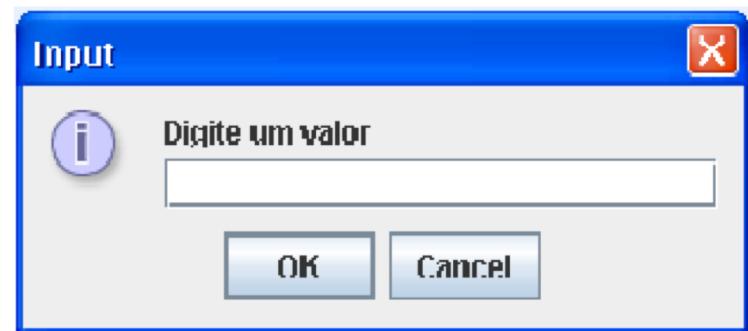
Obs: Retorna um inteiro (0- yes 1-no 2- cancel)

Exemplo InputDialog

Sintaxe: `JOptionPane.showInputDialog`

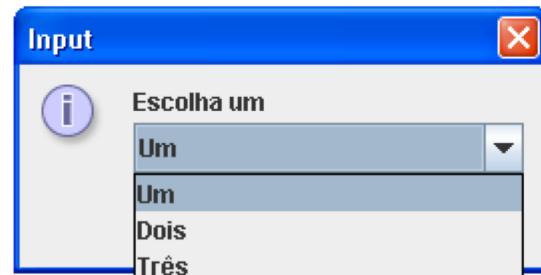
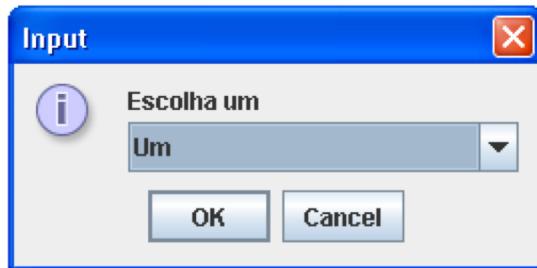
(Componente, mensagem, titulo, tipo da mensagem, ícone,
array de opções, opção default)

```
import javax.swing.JOptionPane;
public class TesteInputDialog {
    public static void main (String [] args)
    {
        String valor="";
        valor=JOptionPane.showInputDialog(null,
            "Digite um valor", "Input",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
}
```



Exemplo InputDialog com ComboBox

```
import javax.swing.JOptionPane;
public class TesteInputDialog {
    public static void main (String [] args)
    {
        Object[] valoresPossiveis= { "Um", "Dois", "Três" };
        Object selectedValue = JOptionPane.showInputDialog(
                null,"Escolha um", "Input",
                JOptionPane.INFORMATION_MESSAGE, null,
                valoresPossiveis, valoresPossiveis[0]);
        System.exit(0);
    }
}
```

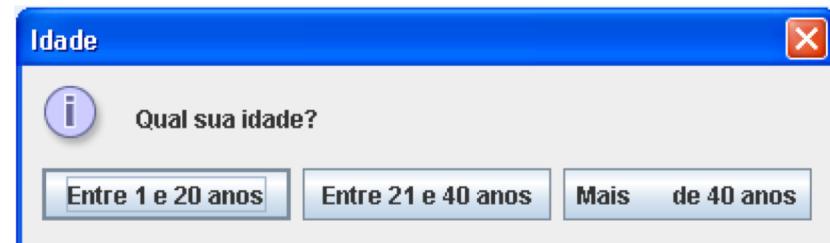


Exemplo JOptionPane

Sintaxe: `JOptionPane.showOptionDialog`

(Componente, mensagem, titulo, tipo dos botões, tipo da mensagem, icone, array de opções, opção default)

```
import javax.swing.JOptionPane;
public class TesteJOptionPane {
    public static void main (String [] args)
    {
        String [] escolha={"Entre 1 e 20 anos",
                           "Entre 21 e 40 anos","Mais de 40 anos"};
        int resp;
        resp=JOptionPane.showOptionDialog(null, "Qual sua idade?",
                                         "Idade", 0,
                                         JOptionPane.INFORMATION_MESSAGE, null,
                                         escolha, escolha[0]);
        System.exit(0);
    }
}
```



Elementos Gráficos

- Os elementos chaves de uma interface gráfica em Java são:
 - ⇒Componentes gráficos (Campos de texto, botões etc.)
 - ⇒Gerenciadores de layout
 - ⇒Processamento de eventos
- Componentes gráficos, tais como campo texto e botões, são elementos que o usuário manipula com o mouse ou com o teclado
- Gerenciadores de layout governam a maneira pela qual os componentes aparecem na tela
- Eventos assinalam ações do usuário consideradas importantes, como o movimento e o *click* de um mouse em cima de um botão, digitação de um campo, seleção de um item do menu,etc.

Programação Dirigida por Eventos

- Programas Java devem responder a eventos gerados por componentes gráficos indicando que ações específicas ocorreram sobre tais componentes
- Uma categoria especial de classes, chamadas *listeners*, que ficam a espera de eventos
- Um programa cliente Java é, geralmente, composto por:
 - o código representando a interface gráfica
 - *listeners* escutando eventos
 - código que respondem a eventos

Diretivas

- As classes abordadas necessitam classes externas, tanto do pacote *swing* quanto do pacote *awt*.
- Diretivas:
 - *import java.awt.*;* -> utilização de classes do pacote AWT, e do acesso à constantes numéricas;
 - *import java.awt.event.*;* -> processamento dos eventos, tais como: click do mouse, ENTER do campo texto, etc;
 - *import javax.swing.*;* -> utilização de classes o pacote *swing*.

Criando aplicações gráficas

- A criação de interfaces gráficas AWT consiste basicamente em:
 - criação (instância) de objetos do tipo **Component** (botões, textos etc.);
 - criação de recipientes ou objetos da classe **Container** (janelas, painéis etc.) para receber os componentes criados;
 - adição dos componentes aos recipientes, com base num sistema de coordenadas (especificando a localização da inserção) ou via utilização de gerentes de layouts que se encarregam de definir a localização e aspecto visual dos componentes inseridos nos recipientes.

Component

- A classe Component representa um objeto gráfico, como botões, campos de textos, choices etc.
- Define o comportamento básico para a maioria dos componentes visuais do pacote AWT (todos os métodos definidos nesta classe estarão disponíveis para todos os componentes visuais do AWT).
- Ex.: método setVisible(boolean visibilidade)
 - botao.setVisible(true); // faz um Button aparecer no Container
 - janela.setVisible(false); // faz um Frame desaparecer

Containers

- Um **container** é um painel de conteúdo.
- Um *container* é uma categoria especial de componente gráfico que pode conter outros componentes ou mesmo outros containers.
- Todos os containers são componentes, mas nem todos os componentes são containers.
- Uma janela e um painel são exemplos de container.
- Cada container possui associado um **gerenciador de layout** para controlar a maneira pela qual seus componentes serão mostrados (tamanho e posição).

Frame

- Um *Frame* é uma janela com uma *barra de título* e uma *borda*.
- A classe *Frame* especializa a classe *Window*, que por sua vez, especializa a classe *Container*.
- A classe *Frame* implementa a interface *MenuContainer*, logo uma frame pode ter uma barra de *menu* associada a ela.
- *Frames* são geralmente usadas para construir aplicações, mas também podem ser usadas com *applets*.

JFrame

Construtor

`JFrame()`

Operação

Constrói uma nova instância de Frame que é inicialmente invisível.

`JFrame(String t)`

Constrói uma nova instância de Frame que é inicialmente invisível, com o título passado como parâmetro.

JFrame

Método

`setTitle(String t)`

Uso

Atribui um título, passado como parâmetro, ao objeto Frame.

`setResizable(boolean b)`

Permite, ou não, que o usuário modifique o tamanho do Frame.

`setVisible(boolean b)`

Torna, ou não, o Frame visível.

`setLocation(int,int)`

Localização do Frame

`setDefaultCloseOperation (int)`

Define a operação a ser executada quando o frame for fechado.

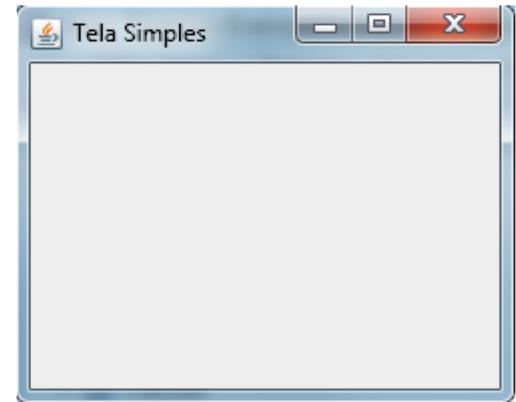
.....

JFrame

- Deve-se usar `pack()` ou `setSize(int,int)` em um Frame antes que ele seja mostrado pela primeira vez. Caso contrário, somente a barra de título da janela aparecerá na tela.
 - ⇒ pack delega para o gerenciador de layout a mudança no tamanho da janela.
 - ⇒ com o gerenciador de layout adquire-se melhor independência de plataforma.

JFrame - exemplo

```
import javax.swing.*;  
public class JFrameBasico extends JFrame {  
  
    JFrameBasico (String titulo) {  
        setTitle(titulo);  
        setSize(250, 200);  
        setVisible(true);  
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
    }  
  
    public static void main(String[] args) {  
        JFrameBasico jfb = new JFrameBasico ("Tela  
        Simples");  
        }  
    }
```



JPanel

- A classe JPanel (painel) é uma sub-classe de JComponent.
- Agrupa componentes para serem posteriormente inseridos em outro container.
- Permitem a criação de layouts sofisticados.
- Podem conter componentes incluindo outros painéis.
- Utiliza o gerente de layout FlowLayout como padrão.

JPanel

Construtor

JPanel()

JPanel(LayoutManager I)

Operação

Constrói uma nova instância de um painel que é inicialmente invisível.

Constrói uma nova instância de um painel contendo um gerente de layout específico.

JPanel

- O exemplo a seguir mostra a criação de um panel que pode ser usado para inserir novos componentes.
- Obs.: lembre-se que um panel é uma subclasse da classe JComponent, logo poderá utilizar o método add(Component c) para inserir algum componente internamente.

JPanel - exemplo

```
import javax.swing.*;  
  
class JFrameBasic01{  
  
    public static void main(String args[]) {  
        JPanel jpanel = new JPanel( );  
        JButton b = new JButton("Botão");  
        jpanel.add(b);  
        JFrame f = new JFrame();  
        f.getContentPane().add(jpanel);  
        f.pack();  
        f.setVisible(true);  
    }  
}
```



Inclusão de novos componentes

JLabel

- Um área onde textos não editáveis e imagens podem ser mostrados
- Rótulos são usados para informações ou instruções textuais na interface gráfica
- Uma vez criado um rótulo, programas raramente modificam seu conteúdo

Sintaxe:

```
JLabel <nome do label>=new JLabel  
(" <texto do label> ", JLabel.<alinhamento>)
```

JLabel - métodos

Construtores

```
public JLabel()  
public JLabel (String s)  
public JLabel (String s, int alignment)  
public JLabel(String text,Icon icon,  
             »int horizontalAlignment)  
public JLabel(Icon icon, int horizontalAlignment)
```

Outros métodos

getText()	Obtém o texto do label
setText()	Especifica o texto do label

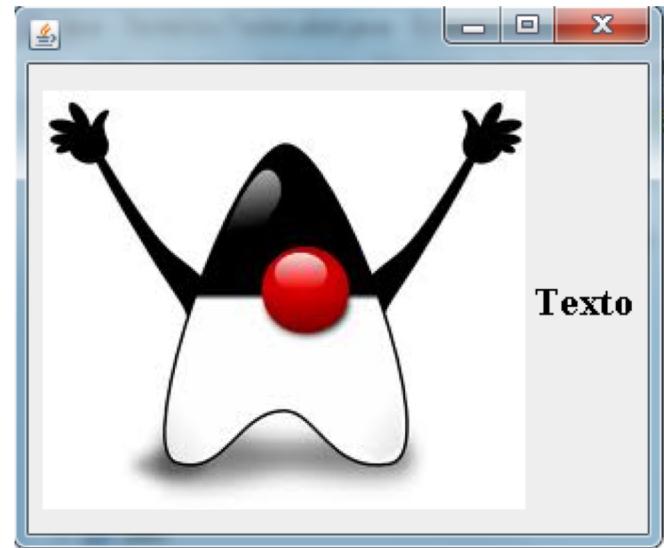
...

Constantes de alinhamento:

- SwingConstants.LEFT
- SwingConstants.CENTER
- SwingConstants.RIGHT

JLabel - exemplo

```
import java.awt.*;
import javax.swing.*;
public class TesteLabel
{
    public static void main(String[] args)
    {
        JFrame fr = new JFrame();
        JLabel lb = new JLabel("Texto",
                              new ImageIcon("duke.gif"),
                              SwingConstants.CENTER);
        lb.setFont(new Font("Serif",Font.BOLD,20));
        lb.setForeground(Color.black);
        fr.getContentPane().add(lb);
        fr.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        fr.pack();
        fr.setVisible(true);
    }
}
```



Botões

- O pacote javax.swing suporta os seguintes tipos de botões:
 - **JButton** Botão comum
 - **JCheckBox** Botões de seleção independentes
 - **JRadioButton** Botões agrupados para seleção
 - **JMenuItem** Item de menu no formato de botão
- JCheckBox ou
JRadioButton.

JButton

É subclasse de *AbstractButton* e *JComponent*.

- **public JButton()**
 - constrói um objeto do tipo Botão sem texto.
- **public JButton(String texto)**
 - constrói um objeto Botão com o texto especificado.
- **public JButton(Icon icon)**
 - constrói um objeto Botão com o ícone especificado.
- **public JButton(String texto, Icon icon)**
 - constrói um objeto Botão com o texto especificado e um ícone.

JButton - métodos

- **getText()**
 - retorna o String representando o texto vinculado ao botão.
- **setText(String s)**
 - define um texto para representar o botão.
- **getIcon()**
 - retorna o ícone vinculado ao botão.
- **setIcon(Icon icon)**
 - define um ícone para ser mostrado pelo botão.
- **setToolTipText(String s)**
 - possibilita atrelar uma mensagem ao botão, quando o ponteiro do mouse estaciona sobre o botão

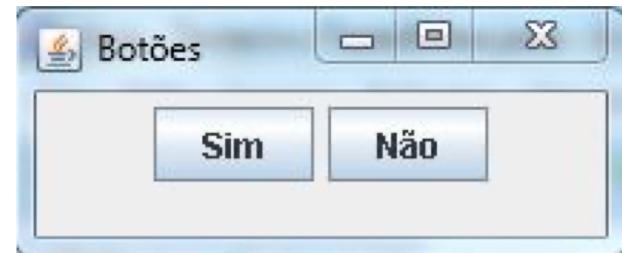
JButton – exemplo

```
import javax.swing.*;
import java.awt.*;
public class TesteBotao {
    private JButton bsim;
    private JButton bnao;
    private JFrame janelaDemo;

    TesteBotao() {
        janelaDemo = new JFrame("Botões");
        janelaDemo.getContentPane().setLayout(new FlowLayout());
        bsim= new JButton("Sim");
        bnao= new JButton("Não");
        bsim.setToolTipText("Pressione Sim para sair");
        janelaDemo.getContentPane().add(bsim);
        janelaDemo.getContentPane().add(bnao);

        janelaDemo.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        janelaDemo.pack();
        janelaDemo.setVisible(true);
    }

    public static void main(String[] args) {
        TesteBotao t = new TesteBotao();
    }
}
```



JTextField

- Um objeto da classe *JTextField* é um campo texto na forma de uma linha na qual textos podem ser digitados pelo usuário através do teclado.
- A classe *JTextField* é uma sub-classe de *JTextComponent*, portanto pode-se aplicar nesta todos os métodos da classe *JTextComponent*.

JTextField - métodos

Construtores

- `public JTextField()`
 - constrói um objeto do tipo campo de texto.
- `public JTextField(int colunas)`
 - constrói um objeto campo de texto vazio com o número especificado de colunas.
- `public JTextField(String texto)`
 - constrói um objeto campo de texto com o texto fornecido no parâmetro texto.
- `public JTextField(String s, int colunas)`
 - constrói um objeto campo de texto com o texto fornecido no parâmetro texto com o número especificado de colunas.

Outros métodos

- `public String getText()` - retorna o que foi digitado para o campo
- `public void setText(String t)` - atribui uma string para o campo

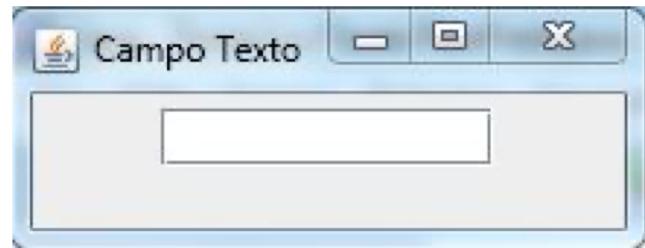
JTextField – exemplo

```
import javax.swing.*;
import java.awt.*;
public class TesteJTextField {
    private JTextField jtfTeste;
    private JFrame janelaDemo;

    public TesteJTextField() {
        String t="";
        janelaDemo = new JFrame("Campo Texto");
        janelaDemo.getContentPane().setLayout(new FlowLayout());
        jtfTeste= new JTextField(10);
        janelaDemo.getContentPane().add(jtfTeste);

        janelaDemo.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        janelaDemo.pack();
        janelaDemo.setVisible(true);
    }

    public static void main(String[] args) {
        TesteJTextField t = new TesteJTextField();
    }
}
```



JPasswordField

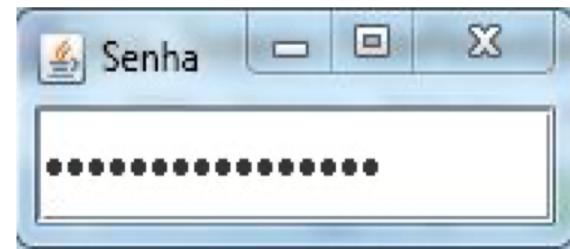
- O objetivo desta classe é incluir caixas de senha nos frames.
- O funcionamento desta classe é praticamente o mesmo da classe *JTextField*. A diferença é que o caracter digitado é substituído por outro caracter para esconder a senha digitada.
- O caracter default é “*”, entretanto qualquer caracter pode ser definido através do método *setEchoChar(char)*.

JPasswordField – exemplo

```
import java.awt.*;
import javax.swing.*;
public class TestePasswd extends JFrame{
    private JPasswordField p1;

    public TestePasswd() {
        setTitle("Senha");
        p1=new JPasswordField();
        getContentPane().add(p1);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        TestePasswd t = new TestePasswd();
    }
}
```



JComboBox

- É herança direta de **JComponent**.
- Um JComboBox é um componente visual que possibilita a manipulação de coleções de objetos permitindo ao usuário selecionar apenas um objeto da coleção. Cada objeto inserido no JComboBox é representado visualmente pela String retornada pelo método `toString()`.

Construtores

- **public JComboBox(Object[] items)**
 - Cria um objeto JComboBox que recebe seus itens de um array de objetos. O objeto JComboBox utilizará o método `toString` de cada objeto do array.
- **public JComboBox()**
 - Cria um objeto JComboBox com um modelo de dados padrão (uma lista de objetos vazia).

JComboBox

Métodos

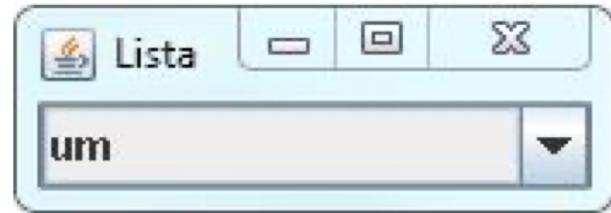
- **public Object getSelectedItem()**
 - Retorna o item correntemente selecionado.
 - **public int getSelectedIndex()**
 - Retorna o posicional do item correntemente selecionado.
 - **public void addItem(Object anObject)**
 - Adiciona um elemento na lista padrão (usado quando o JComboBox é criado com construtor vazio).
 - **public void insertItemAt(Object anObject, int index)**
 - Insere um elemento na lista padrão na posição informada pelo índice (usado quando o JComboBox é criado com construtor vazio).
 - **public Object getItemAt(int index)**
 - Retorna o objeto referente ao elemento posicionado no índice fornecido.
-

JComboBox – exemplo

```
import java.awt.*;
import javax.swing.*;
public class TesteJComboBox extends JFrame{
    private JComboBox lista;

    TesteJComboBox () {
        String[] data = {"um", "dois", "três", "quatro"};
        setTitle("Lista");
        lista=new JComboBox(data);
        JScrollPane sp=new JScrollPane(lista);
        getContentPane().add(sp);
        getContentPane().add(lista);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        TesteJComboBox t = new TesteJComboBox();
    }
}
```



JCheckBox

- É subclasse de JToggleButton, AbstractButton e JComponent.
- Um **JCheckBox** pode ter dois estados: ligado ou desligado.
- Um conjunto de *checkboxes* podem ser agrupados através da classe ButtonGroup. Neste caso, a seleção é mutuamente exclusiva.

JCheckBox

Construtores

Uso

JCheckBox(String label) Cria um Checkbox desligado, com o label especificado.

JCheckBox(String label,boolean state)

Cria um Checkbox ligado ou desligado, de acordo com a variável booleana state e com o label especificado.

JCheckBox(Icon icon, boolean state)

Cria um Checkbox ligado ou desligado, de acordo com a variável booleana state, com o ícone especificado.

Métodos

isSelected()

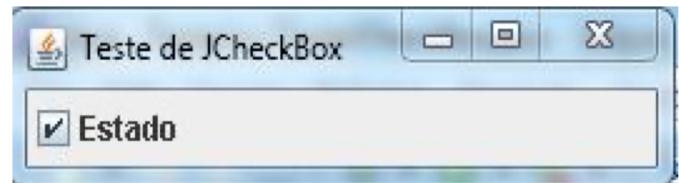
Obtém o estado do checkbox, retornando verdadeiro ou falso

setSelected(boolean b)

Especifica o estado do checkbox

JCheckBox - exemplo

```
import javax.swing.*;  
class TesteJCheckBox extends JFrame{  
    private JCheckBox jcb;  
  
    TesteJCheckBox(String titulo){  
        setTitle(titulo);  
        String estado="";  
        jcb=new JCheckBox("Estado");  
        jcb.setSelected(true);  
        if(jcb.isSelected())          estado="ligado";  
        else                          estado="desligado";  
        getContentPane().add(jcb);  
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
        setVisible(true);  
        pack();  
    }  
  
    public static void main(String args[]){  
        TesteJCheckBox tcb= new TesteJCheckBox("Teste de  
JCheckBox");  
    }  
}
```



JRadioButton

- É subclasse de JToggleButton, AbstractButton e JComponent.
- Pode ter dois estados : ligado e desligado.
- Podem ser agrupados pela classe *ButtonGroup*. Neste caso, a seleção é mutuamente exclusiva.

JRadioButton

```
import javax.swing.*;
class TesteRadioButton extends JFrame{
    private JRadioButton jrbm;
    private JRadioButton jrbf;

    TesteRadioButton(String titulo){
        setTitle(titulo);
        JPanel painel=new JPanel();
        jrbm=new JRadioButton("Masculino");
        jrbf=new JRadioButton("Feminino");
        ButtonGroup grupo = new ButtonGroup ();
        grupo.add(jrbm);
        grupo.add(jrbf);
        painel.add(jrbm);
        painel.add(jrbf);
        getContentPane().add(painel);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setVisible(true);
        pack();
    }

    public static void main(String args[]){
        TesteRadioButton trb= new TesteRadioButton("Teste de
JRadioButton");
    }
}
```



JTextArea

- Um objeto da classe *JTextArea* é um campo texto na forma de várias linhas nas quais textos podem ser digitados pelo usuário através do teclado.
- A classe *JTextArea* é também uma sub-classe de *JTextComponent*, portanto pode-se aplicar nesta todos os métodos da classe *JTextComponent*.

JTextArea - construtores

public JTextArea()

- constrói um objeto do tipo campo de texto com possibilidade para várias linhas.

public JTextArea(int linhas, int colunas)

- constrói um objeto JTextArea vazio com o número especificado de colunas e linhas.

public JTextArea(String texto)

- constrói um objeto JTextArea com o texto fornecido no parâmetro texto.

public JTextArea(String texto , int linhas, int col)

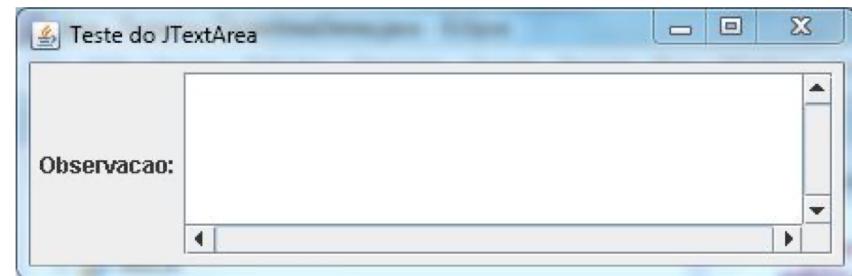
- constrói um objeto JTextArea com o texto especificado e número especificado de colunas e linhas.

JTextArea - métodos

- **public String getText()**
 - retorna o que foi digitado para o campo
- **public void setText(String t)**
 - atribui uma string para o campo
- **public void setColumns(int c)**
 - define o número de colunas visíveis para o objeto JTextArea.
- **public int getColumns()**
 - devolve o número de colunas visíveis já definidas para o objeto JTextArea.
- **public void setRows(int r)**
 - define o número de linhas visíveis para o objeto JTextArea.
- **public int getRows()**
 - devolve o número de linhas visíveis já definidas para o objeto JTextArea.
- **append(String str)**
 - adiciona o texto no final do JTextArea.
- **insert(String str, int pos)**
 - insere o texto na posição especificada.
-

JTextArea - exemplo

```
import java.awt.*;
import javax.swing.*;
public class TextAreaDemo {
    public TextAreaDemo() {
        JLabel lNome = new JLabel("Observação:");
        JTextArea tArea = new JTextArea("", 5, 30);
        JScrollPane sp = new JScrollPane(tArea,
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        JPanel painel = new JPanel();
        JFrame janela = new JFrame("Teste do JTextArea");
        painel.add(lNome );
        painel.add(sp );
        janela.getContentPane().add(painel);
        janela.pack();
        janela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        janela.setVisible(true);
    }
    public static void main(String[] args) {
        TextAreaDemo t = new TextAreaDemo();
    }
}
```



JMenuBar

É subclasse de JComponent. O layout default é o BoxLayout.

JMenuBar()

Constrói um objeto do tipo barra de menu.

add(JMenu c)

Adiciona o menu especificado ao final da barra de menu.

Para atribuir a barra de menu ao JFrame desejado, basta utilizar o método da janela:

void **setJMenuBar(JMenuBar menubar)**

JMenu

É subclasse de JMenuItem. Utilizada para criar os itens principais de menu ou submenus.

JMenu()

Cria um menu principal ou submenu.

Para atribuir um texto, utilize **setText(String s)**.

JMenu(String s)

Cria um menu principal ou submenu com o texto especificado.

Para atribuir um ícone, utilize **setIcon(Icon i)**.

add(String s)

Cria um item de menu com o texto especificado e o adiciona ao final do menu.

add(JMenuItem j)

Adiciona o item de menu especificado ao final da barra de menu.

addSeparator()

Adiciona um separador como próximo (ou último) item do menu.

JMenuItem

É subclasse de JAbstractButton, ou seja, os itens de menu são simplesmente botões.

Construtores

JMenuItem()

Cria um item de menu.

JMenuItem(String s)

Cria um item de menu com o texto especificado.

JMenuItem(String s, Icon i)

Cria um item de menu com o texto e ícone especificados.

JMenuItem(String s, int mnemonic)

Cria um item de menu com o texto e mnemônico especificados.

JMenuItem

Alguns métodos importantes :

void setMnemonic (int mnemônico)

- Herdado de JAbstractButton, atribui uma tecla de acesso ao item de menu.

ex.: itemDeMenu.setMnemonic(72); // char N.

void setAccelerator (KeyStroke keyStroke)

- Atribui uma tecla de atalho para execução direta do item de menu.

ex.: itemDeMenu.setAccelerator(
KeyStroke.getKeyStroke(

ActionEvent.ALT_MASK));

KeyEvent.VK_B,

***resultará na tecla de atalho ALT_B.**

Passos para criar um menu

- 1.** Criar a `JMenuBar()` e atribuí-la ao `JFrame` desejado.
- 2.** Criar os menus principais ou submenus do tipo `JMenu()` e adicioná-los à barra de menu (`JMenuBar`) ou a um menu (`JMenu`) como um novo item.
- 3.** Criar os itens de menu do tipo `JMenuItem()` e adicioná-los ao menu principal ou ao submenu correspondente (`JMenu`).
- 4.** Se houver subitens de menu, repetir os passos 2 e 3.

JMenu - exemplo

```
import javax.swing.*;
public class TesteMenuBar extends JFrame {
    private JMenuBar jmbBarraMenu;
    private JMenu jmArquivo;
    private JMenuItem jmiNovo;
    private JMenuItem jmiAbrir;
    private JMenuItem jmiFechar;
    private JMenuItem jmiSalvar;
    private JMenuItem jmiSair;
    public TesteMenuBar() {
        super("Teste de Menu");
        jmbBarraMenu=new JMenuBar();
        setJMenuBar(jmbBarraMenu);
        jmArquivo=new JMenu("Arquivo");
        jmbBarraMenu.add(jmArquivo);
        jmiNovo=new JMenuItem("Novo");
        jmArquivo.add(jmiNovo);
        jmiAbrir=new JMenuItem("Abrir",new
ImageIcon("open.gif"));
        jmArquivo.add(jmiAbrir);
        jmiAbrir.setMnemonic(65);
        jmiFechar=new JMenuItem("Fechar");
        jmArquivo.add(jmiFechar);
        jmiSalvar=new JMenuItem("Salvar");
        jmArquivo.add(jmiSalvar);
        jmiSair=new JMenuItem("Sair");
        jmArquivo.add(jmiSair);
        pack(); setVisible(true);
    }
    public static void main(String[] args) {
        TesteMenuBar t = new TesteMenuBar();
    }
}
```



Gerenciadores de Layout

- Importância da aparência e estética da aplicação
- Embora seja possível posicionar objetos em coordenadas fixas (x, y), deve-se evitar essa prática em Java.
- Isso causa problemas de portabilidade: outras JVMs, GUIs e *Window Managers* terão fontes e outros elementos em tamanhos e proporções diferentes.
- Há 6 gerenciadores de layout pré-definidos no pacote `java.awt.*`:
 - **FlowLayout**
 - **GridLayout**
 - **BorderLayout**
 - **CardLayout**
 - **GridBagLayout**
 - **BoxLayout**

Gerenciadores de Layout (cont.)

- Implicitamente, cada *container* possui um gerenciador de layout default associado.
- Um programador pode também criar novos gerenciadores de layout.

Escolhendo um gerenciador de layout

- Se você precisa mostrar alguns componentes utilizando os seus tamanhos naturais em uma linha, então use o **FlowLayout**.
- Se você precisa mostrar componentes de mesmo tamanho em linhas e colunas, então use o **GridLayout**.
- Se você precisa mostrar painéis alternativamente, então use o **CardLayout**.
- Se você precisa mostrar componentes e espalhá-los utilizando todo espaço disponível, então utilize o **BorderLayout** ou **GridBagLayout**.

FlowLayout

- O FlowLayout é o layout default em [painéis](#).
- O FlowLayout se constitui no mais simples gerenciador de layout da linguagem Java. Nele os componentes são dispostos da esquerda para a direita na ordem em que aparecerem, isto é, na ordem em que os componentes são adicionados à janela pelo método **add**. Quando não existe mais espaço em uma linha é criada outra linha abaixo desta e este critério é usado em toda a janela. A distribuição dos objetos por meio do FlowLayout pode ser comparada à distribuição das palavras que um editor de texto qualquer faz.

FlowLayout

- **Sintaxe utilizada para a definição do layout:**
`getContentPane().setLayout(new FlowLayout([Alinhamento,
Espaçamento_horizontal, Espaçamento_vertical]))`
- **Alinhamento** - refere-se ao alinhamento que os componentes assumirão no momento em que forem inseridos no container, podendo assumir os seguintes valores inteiros: 0=esquerda, 1=centralizado, 2=direita. O alinhamento é opcional, e quando não especificado assume o valor 0.
- **Espaçamento_horizontal** - refere-se à distância que será dada entre os objetos inseridos na mesma linha do container. O espaçamento horizontal é opcional e se não for especificado é considerado com 5 unidades.
- **Espaçamento_vertical** - se refere à distância que será dada entre as linhas do container onde os objetos estão inseridos. O espaçamento é opcional e se não for especificado é considerado com 5 unidades.

FlowLayout

- **Construtores** **Função**
- `FlowLayout()` Cria um layout padrão: alinhamento centralizado e espaçamento horizontal e vertical com 5 unidades.
- `FlowLayout(int)` Cria um layout com alinhamento fornecido e espaçamentos com cinco unidades.
- `FlowLayout(int,int,int)` Cria um layout com o alinhamento,

FlowLayout - exemplo

```
import javax.swing.*;
import java.awt.*;
public class TesteFlowLayout extends JFrame {

    private JButton b1;
    private JButton b2;
    private JButton b3;
    private JButton b4;
    private JButton b5;
    public TesteFlowLayout() {

        super("Teste do FlowLayout");
        b1=new JButton("Botão 1");
        b2=new JButton("Botão 2");
        b3=new JButton("Botão 3");
        b4=new JButton("Botão 4");
        b5=new JButton("Botão 5");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(b1);
        getContentPane().add(b2);
        getContentPane().add(b3);
        getContentPane().add(b4);
        getContentPane().add(b5);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        TesteFlowLayout t = new TesteFlowLayout();
    }
}
```



GridLayout

- O GridLayout é um gerenciador que divide um container num conjunto de células espalhadas numa grade retangular, de maneira que todas as células possuam a mesma direção. Com o GridLayout uma janela pode ser dividida em linhas e colunas de acordo com os argumentos especificados no momento de sua criação. Os componentes são dispostos na ordem em que aparecem, sendo inseridos na grade da esquerda para a direita e de cima para baixo.

GridLayout

- **Sintaxe utilizada para a definição do layout:**

```
getContentPane().setLayout(new GridLayout([Número_de_linhas,  
Número_de_colunas, Espaçamento_horizontal, Espaçamento_vertical]))
```
- **Número_de_linhas** - refere-se à quantidade de linhas que a grade irá conter. Se não for especificado, assume o valor 1.
- **Número_de_colunas** - refere-se à quantidade de linhas que a grade irá conter. Se não for especificado, assume o valor 1.
- **Espaçamento_horizontal** - refere-se à distância que será dada entre os objetos inseridos na mesma linha do container. O espaçamento horizontal é opcional e se não for especificado é considerado com 5 unidades.
- **Espaçamento_vertical** - se refere à distância que será dada entre as linhas do container onde os objetos estão inseridos. O espaçamento é opcional e se não for especificado é considerado com 5 unidades.

GridLayout

Construtores

GridLayout()
linha e uma coluna

GridLayout(int,int)
linhas e
colunas especificados.

GridLayout(int,int,int,int)
linhas e
colunas especificados e com os
especificados.

Função

Cria um layout com uma

Cria um layout com o número de

Cria um layout com o número de
espaçamentos

GridLayout - exemplo

```
import javax.swing.*;
import java.awt.*;
public class TesteGridLayout extends JFrame {

    private JButton b1;
    private JButton b2;
    private JButton b3;
    private JButton b4;
    private JButton b5;
    public TesteGridLayout () {

        super("Teste do GridLayout");
        b1=new JButton("Botão 1");
        b2=new JButton("Botão 2");
        b3=new JButton("Botão 3");
        b4=new JButton("Botão 4");
        b5=new JButton("Botão 5");
        getContentPane().setLayout(new GridLayout(3,5,5,5));
        getContentPane().add(b1);
        getContentPane().add(b2);
        getContentPane().add(b3);
        getContentPane().add(b4);
        getContentPane().add(b5);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        TesteGridLayout t = new TesteGridLayout ();
    }
}
```



BorderLayout

- O BorderLayout é um gerenciador que divide um container em cinco regiões distintas: north (região superior), south (região inferior), west (região à esquerda), east (região à direita) e center (região central). Diferentemente dos gerenciadores flowLayout e GridLayout, a ordem com que os componentes são inseridos é irrelevantes, uma vez que o container é dividido em regiões fixas. Em cada região cabe apenas um componente, ou seja, como o container é dividido em cinco regiões, apenas cinco componentes podem ser inseridos nesse layout.

BorderLayout

- **Sintaxe utilizada para a definição do layout:**

```
getContentPane().setLayout(new BorderLayout([Espaçamento_horizontal,  
Espaçamento_vertical]))
```

- **Espaçamento_horizontal** - refere-se à distância que será dada entre os objetos inseridos na mesma linha do container. O espaçamento horizontal é opcional e se não for especificado é considerado com 5 unidades.
- **Espaçamento_vertical** - se refere à distância que será dada entre as linhas do container onde os objetos estão inseridos. O espaçamento é opcional e se não for especificado é considerado com 5 unidades.

- **Construtores**

BorderLayout()
entre as regiões

BorderLayout(int,int)
entre as regiões

Função

Cria um layout sem espaçamento

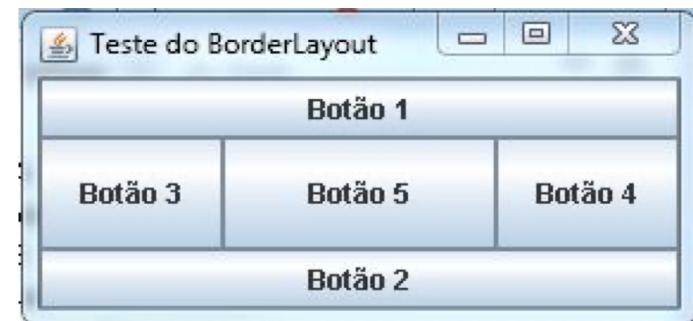
Cria um layout com espaçamento horizontal e
vertical

BorderLayout - exemplo

```
import javax.swing.*;
import java.awt.*;
public class TesteBrderLayout extends JFrame {

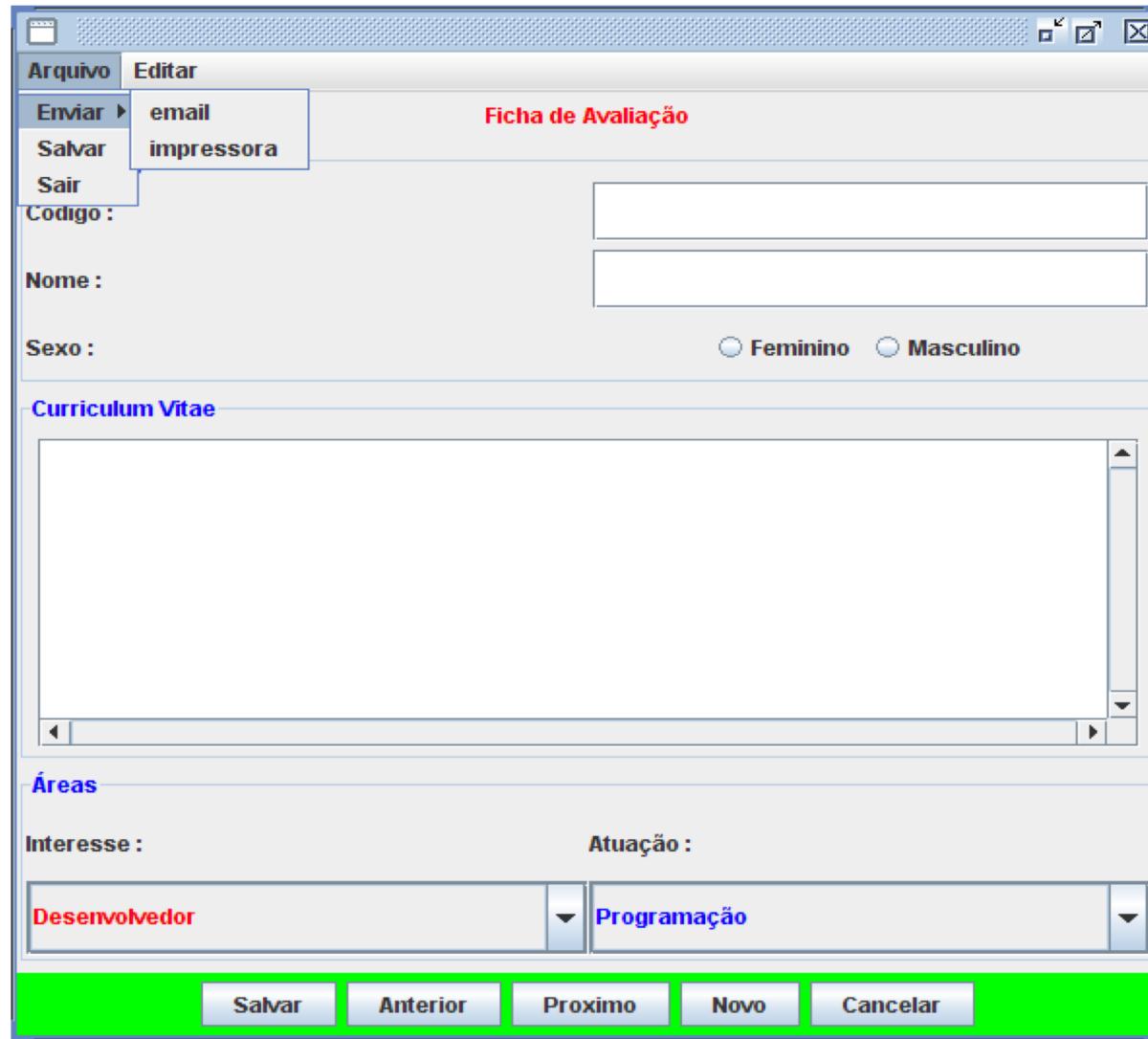
    private JButton b1;
    private JButton b2;
    private JButton b3;
    private JButton b4;
    private JButton b5;
    public TesteBrderLayout() {

        super("Teste do BorderLayout");
        b1=new JButton("Botão 1");
        b2=new JButton("Botão 2");
        b3=new JButton("Botão 3");
        b4=new JButton("Botão 4");
        b5=new JButton("Botão 5");
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add("North",b1);
        getContentPane().add("South",b2);
        getContentPane().add("West",b3);
        getContentPane().add("East",b4);
        getContentPane().add("Center",b5);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        TesteBrderLayout t = new TesteBrderLayout();
    }
}
```



Exercício

- Usando as classes do pacote swing, faça um programa em Java que tenha a seguinte interface:



Tratamento de Eventos

A interface gráfica em Java é orientada a eventos:

- Cada vez que um usuário clica em um botão, seleciona um item em uma lista, ou pressiona uma tecla, o sistema operacional gera um evento;
- Se uma aplicação está interessada em tratar um evento específico (por exemplo, clique em um botão), deve solicitar ao sistema para “**escutar**” o evento;
- Se a aplicação não está interessada, seu processamento continua de forma normal. É importante observar que a aplicação não espera pela ocorrência de eventos: isso é controlado pelo sistema;
- Cada componente suporta vários tipos de eventos;
- Para que um componente ou *container* possa “**escutar**” eventos, é preciso registrar um *listener*;
-

Listeners são classes criadas especificamente para o tratamento de eventos

Tratamento de Eventos

- São eventos comuns:
 - **ActionEvent** : Usuário clica em um botão, pressiona enter dentro de um textfield ou escolhe um item de menu
 - **ComponentEvent** : Componente fica visível, é adicionado a um container
 - **FocusEvent** : Componente recebe o foco do teclado
 - **ItemEvent** : Seleção de uma tabela ou lista é modificada
 - **WindowEvent** : Usuário fecha, minimiza, maximiza, etc., uma janela
 - **MouseEvent** : Usuário pressiona entra ou sai com o mouse de dentro um componente
 - **MouseMotionEvent** : Usuário move o mouse sobre um componente

Tratamento de Eventos

O mecanismo de evento tem três partes significativas e diferentes:

- **A origem do evento** é o componente com o qual o usuário interage para gerar o evento, um botão, por exemplo;
- **O objeto do evento** é o objeto que encapsula as informações sobre o evento, é na realidade o evento em si;
- **O listener, ouvinte** é o objeto que é notificado pelo evento, recebe um objeto evento quando notificado sobre ele.

O processamento do evento é delegado ao ouvinte de eventos no aplicativo, ou melhor, quando ocorre um evento existe um mecanismo de aviso (**dispatch**) para os ouvintes daquele evento. Este **dispatch** é na realidade uma delegação para que o ouvinte cuide do tratamento de evento.

Tratamento de Eventos

- Para cada tipo de evento, ou melhor, objeto evento, existe uma interface ouvinte do evento correspondente. Esta interface é a encarregada deste tratamento.
- Cada interface ouvinte especifica um ou mais métodos de tratamento do evento. Estes métodos devem ser declarados na classe que implementa a interface.

Observação : Lembre-se que uma interface declara métodos que devem ser codificados pelas classes que a implementam.

- Cada tipo de evento tem uma ou mais interfaces ouvintes de eventos.
- São interfaces comuns entre outras::
 - ActionListener
 - ComponentListener
 - FocusListener
 - ItemListener
 - MouseListener
 - TextListener
 - WindowListener

Tratamento de Eventos

- Três situações devem ser observadas para o tratamento de eventos:
 - Todo componente deve registrar que haverá um evento associado a ele e que será motivo de programação específica;
 - Sempre deverá ter um ouvinte (listener) que responderá a um evento;
 - Cabe a interface do ouvinte especificar um método de tratamento do evento;
- Todo ouvinte tem uma lista de eventos aos quais deverá reagir. O programador deve registrar o evento na lista dos eventos que o ouvinte deve reagir e implementas os métodos de tratamento de eventos especificados pelas interfaces dos ouvintes.
- Quando um evento ocorre, o componente recebe da JVM um identificador (um número) que especifica o tipo de evento. Este ID é usado para decidir como será o dispatch, qual o método chamar para cada tipo de ouvinte.
 - Por exemplo, para um ActionEvent o evento é despachado para o método actionPerformed;

Tratamento de Eventos

Interface	Registro do Ouvinte	Métodos
• ActionListener addActionListener()		actionPerformed(ActionEvent)
• AdjustmentListener		adjustmentValueChanged(AdjustmentEvent)
• ComponentListener	componentHidden(ComponentEvent)	addAdjustmentListener() addComponentListener()
		componentMoved(ComponentEvent)
		componentResized(ComponentEvent)
		componentShown(ComponentEvent)
• FocusListener addFocusListener()		focusGained(FocusEvent)
		focusLost(FocusEvent)
• ItemListener addItemListener()		itemStateChanged(ItemEvent)
• MouseListener addMouseListener()		mouseClicked(MouseEvent)
		mouseEntered(MouseEvent)
		mouseExited(MouseEvent)
		mousePressed(MouseEvent)
		mouseReleased(MouseEvent)
• MouseMotionListener addMouseMotionListener()		mouseDragged(MouseMotionEvent)
		mouseMoved(MouseMotionEvent)
• TextListener addTextListener()		textValueChanged(TextEvent)
• WindowListener addWindowListener()		windowClosed(WindowEvent e)
		windowIconified(WindowEvent e)
		windowOpened(WindowEvent e)

Tratamento de Eventos

Exemplo : JTextField e JButton

```
public class Eventos1 extends JFrame {
    JLabel lb1;
    JTextField nome;
    JButton b1;
    Eventos1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout( new GridLayout(0,3,5,5) );
        TrataEvento trata = new TrataEvento();
        lb1 = new JLabel("Nome :");
        nome = new JTextField (10);
        b1 = new JButton("Botão 1");
        nome.addActionListener(trata);
        b1.addActionListener(trata);
        c.add(lb1);
        c.add(nome);
        c.add(b1);
        pack();
        setVisible(true);
    }
    public static void main( String args[] ) {
        Eventos1 janela = new Eventos1();
    }
    private class TrataEvento implements ActionListener {
        public void actionPerformed(ActionEvent evento) {
            if (evento.getSource() == b1)
                JOptionPane.showMessageDialog(null,"Apertou o botao B1");
            if (evento.getSource() == nome)
                JOptionPane.showMessageDialog(null,"Deu Enter no nome " + nome.getText());
        }
    }
}
```

Tratamento de Eventos

Exemplo : Detectando a Posicao do Mouse

```
public class mouse extends JFrame {
    private JLabel status;
    public mouse() {
        super("Mouse Demo");
        status = new JLabel();
        getContentPane().add(status, BorderLayout.SOUTH);
        Posicao ml = new Posicao();
        this.addMouseListener(ml);
        setSize(300,100);
        setVisible(true);
    }//construtor
    //classe interna
    private class Posicao implements MouseListener {
        public void mouseClicked(MouseEvent e) {
            status.setText("clicou em "+ e.getX()+ "; "+ e.getY());
        }
        public void mousePressed(MouseEvent e) {
            status.setText("pressionou em "+ e.getX()+ "; "+ e.getY());
        }
        public void mouseReleased(MouseEvent e) {
            status.setText("soltou em "+ e.getX()+ "; "+ e.getY());
        }
        public void mouseEntered(MouseEvent e) {
            JOptionPane.showMessageDialog(null,"Mouse na janela");
        }
        public void mouseExited(MouseEvent e) {
            status.setText("Mouse fora da janela");
        }
    }//interna
    public static void main(String[] args) {
        mouse m = new mouse();
    }
}
```

Tratamento de Eventos

Exemplo : JCheckBox

```
public class EventosCheckBox extends JFrame {
    JCheckBox bitalico,bnegrito;
    JTextField nome;
    EventosCheckBox(){
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout( new GridLayout(0,3,5,5) );
        TrataEvento trata = new TrataEvento();
        nome = new JTextField (10);
        nome.setFont(new Font("Arial",Font.PLAIN,14));
        bitalico = new JCheckBox ("Itálico");
        bnegrito = new JCheckBox("Negrito");
        bnegrito.addItemListener(trata);
        bitalico.addItemListener(trata);
        c.add(nome); c.add(bnegrito); c.add(bitalico);
        pack();
        setVisible(true);
    }
    public static void main( String args[] ) {
        EventosCheckBox janela = new EventosCheckBox();
    }
    private class TrataEvento implements ItemListener {
        private int valNeg = Font.PLAIN, valIta = Font.PLAIN;
        public void itemStateChanged(ItemEvent evento) {
            if (evento.getSource() == bnegrito)
                if (evento.getStateChange() == ItemEvent.SELECTED)
                    valNeg = Font.BOLD;
                else
                    valNeg = Font.PLAIN;
            if (evento.getSource() == bitalico)
                if (evento.getStateChange() == ItemEvent.SELECTED)
                    valIta = Font.ITALIC;
                else
                    valIta = Font.PLAIN;
            nome.setFont(new Font("Arial", (valNeg + valIta) , 14));
        }
    }
}
```

Tratamento de Eventos

Exemplo : JComboBox

```
public class EventosCombo extends JFrame {
    String [] nomes = {"Blue", "Green", "Red"};
    Color [] cores = {Color.blue, Color.green, Color.red};
    JComboBox lista;
    Container c;
    EventosCombo() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c = getContentPane();
        c.setLayout( new FlowLayout());
        c.setBackground(Color.blue);
        TrataEvento trata = new TrataEvento();
        lista = new JComboBox (nomes);
        lista.addItemListener(trata);
        c.add(lista);
        pack();
        setVisible(true);
    }
    public static void main( String args[] ) {
        EventosCombo janela = new EventosCombo();
    }

    private class TrataEvento implements ItemListener {
        public void itemStateChanged(ItemEvent evento) {
            if (evento.getStateChange()== ItemEvent.SELECTED)
                c.setBackground(cores[lista.getSelectedIndex()]);
        }
    }
}
```

Tratamento de Eventos

Exercícios:

1 – Calculadora com 3 campos JTextField, sendo um deles o de resultado e 4 botões “+”, “-”, “*”, “/”.

2 – Crie um programa de cadastro de funcionários :

Nome, matricula, idade, sexo, cargo e departamento.

Bibliografia

- DEITEL,H.M.; DEITEL, P.J. Java: Como Programar, 8^a Edição, Bookman, 2010.